UNIFORM RANDOM NUMBER GENERATION

Pierre L'Ecuyer

Département d'informatique et de recherche opérationnelle,

Université de Montréal,

C.P. 6128, Succ. Centre-Ville, Montréal, H3C 3J7, Canada.

email: lecuyer@iro.umontreal.ca

http://www.iro.umontreal.ca/~lecuyer

I INTRODUCTION

Random numbers generators (RNGs) can be found in most software libraries or systems. They are a key ingredient for several types of computations, such as for Monte Carlo simulation, statistical methods, probabilistic algorithms in different areas, computer games, cryptography, casino machines, and so on. From the user's point of view, the output of a RNG is usually considered as a sequence of independent and identically distributed (i.i.d.) random variables, often with the uniform distribution over the interval $[0, 1]$. The RNGs should at least *appear* to behave as such. In reality, however, they are simple computer programs whose behavior is completely deterministic after their initial state is chosen. Hence the name *pseudorandom*, often employed. Attempts have been made to construct *truly random* generators based on physical devices such as noise diodes, gamma ray counters, and so on, but these remain largely impractical and unreliable. See [38] for a discussion.

We define a *random number generator* (see [20, 21]) as a structure $\mathcal{G} = (S, \mu, T, U, G)$, where S is a finite set of *states*, μ is a probability distribution on S used to select the *initial state* (or *seed*) s_0, the mapping $T : S \to S$ is the *transition function*, U is a finite set of *output* symbols, and $G : S \to U$ is the *output function*.

The state of the generator starts at s_0 and evolves according to the recurrence $s_n = T(s_{n-1})$, for $n \geq 1$. The *output* at step n is $u_n = G(s_n)$. These u_n are the so-called *random numbers*. Note that the number of states in S can be much larger than the number of distinct output values, so different states can be mapped to the same output, in general. Since S is finite, the generator will eventually return to a state already visited, i.e., $s_{i+j} = s_i$ for some $i \geq 0$ and $j > 0$. Then, $s_{n+j} = s_n$ and $u_{n+j} = u_n$ for all $n \geq i$. The smallest $j > 0$ and smallest $i \geq 0$ for which this happens are called the *period* length ρ, and the *transient* length τ, respectively. If $\tau = 0$, the sequence is called *purely periodic.* Of course, ρ cannot exceed the number of states $|S|$. Well-designed generators have their ρ close to this upper bound.

If the seed is chosen randomly, e.g., uniformly from S, using external randomness such as picking balls from a box or throwing a dice, then the generator can be viewed as an *extensor* of randomness, which stretches a short random seed into a longer sequence of random-looking numbers.

To add further "real randomness" to the sequence and mix-up its regular mathematical structure, one may combine it with physical noise. For example, Marsaglia [38] has produced a CDROM containing 4.8 billion random bits, by combining the output of some random number generators with various sources of noise such as music, pictures, or physical devices, via bitwise exclusive-or. This is interesting, but some applications require much more random numbers than provided on this disk, and reading numbers from a CDROM or from a file does not match the speed and convenience of a good RNG.

To simulate random variables from different probability distributions such as the normal, exponential, binomial, geometric, and so on, or to simulate more complicated random objects, one would transform the output of a RNG in the appropriate way. See, for example, [2, 9, 19]. These transformations are not always simple. They usually assume the availability of a uniform RNG.

Henceforth, we assume that the generator's purpose is to *simulate* i.i.d. $U(0,1)$ random variables (uniform over the interval $[0,1]$). The goal is that no user could easily distinguish between the output sequence of our dream generator and a sequence of i.i.d. $U(0,1)$ random variables, apart from the finite precision. A necessary (but not sufficient) condition is a huge period length, long enough to make sure that no one can ever exhaust it. The RNG must also have good statistical behavior and, in certain contexts such as in cryptography, unpredictability is a major issue as well. From the theoretical viewpoint, unpredictability turns out to be asymptotically equivalent to uniformity and independence. These properties can be tested empirically, but more importantly, should be supported by theoretical analysis. Efficiency is also an issue. For certain types of applications such as simulations in statistical mechanics, the RNG eats up much of the computing time, which is often hours or days on supercomputers. The amount of memory required can be an issue when several generators are used in parallel, which happens frequently in simulation [19].

Repeating exactly the same sequence of random numbers is needed, e.g., for program verification or to compare competing systems via simulation under similar conditions [2, 19]. Repeatability of the sequences is a major advantage of RNGs over physical devices. Ideally, RNG implementations should be *portable*, i.e., work without change and produce the same sequence across most compilers and computers. A single RNG is often split into several virtual generators simply by partitioning its sequence into long disjoint substreams. One then needs fast algorithms to compute the state $s_{n+\nu}$ for any large ν, given s_n. Efficient methods to do this are known for several types of generators, including most of the linear ones (see, e.g., [21] and Section III).

RNGs should be designed on the basis of a sound mathematical analysis of their structural properties. This is sometimes called *theoretical testing*. For example, the parameters of linear congruential or multiple recursive generators are chosen based on a lattice structure analysis called the *spectral test*, which we explain later. Once a generator is constructed, one can apply *empirical statistical tests* to it, to try catching up significant statistical defects. These theoretical and empirical analyses remain heuristic, in the sense that they can never prove that a given generator is fully reliable for any simulation. The best they can do is improve our confidence in the RNG.

The rest of this article is organized as follows. The next section describes theoretical frameworks and criteria for discriminating among generators, and approximate to some extent the dream generator. Section III covers the linear congruential methods based on integer arithmetic and the associated lattice properties. Section IV discusses linear methods based on digital expansions, and some equidistribution properties. Section V is devoted to methods based on nonlinear recurrences, such as inversive congruential generators and other methods stemming from cryptology. Statistical testing is discussed in Section VI, while Section VII gives pointers to well-tested implementations.

II FRAMEWORKS FOR PSEUDORANDOM SEQUENCES

A Unpredictability and computational complexity

A first theoretical framework for pseudo-RNGs is based on asymptotic analysis and *computational complexity theory*. Consider a *family* of generators, $\{\mathcal{G}_k, k = 1, 2, \ldots\}$, where $k = \log_2 |S|$, the number of bits to represent the state. Assume that T and G can be computed in time polynomial in k and consider only the family of statistical tests $\mathcal{T}_p$ taking a time polynomial in k (the tests taking $O(|S|)$ time, for example, are excluded). The family $\{\mathcal{G}_k\}$ is called *polynomial-time perfect* if, for any test in $\mathcal{T}_p$ trying to distinguish the output sequence of the generator from an infinite sequence of i.i.d. $U(0,1)$ random variables, the probability that the test makes the right guess does not exceed $1/2 + e^{-k\epsilon}$, for some constant $\epsilon > 0$. It has been shown that the following requirement (among others) is equivalent: No polynomial-time algorithm can predict any given bit of u_n with probability of success larger than $1/2 + e^{-k\epsilon}$, after observing $u_0, \ldots, u_{n-1}$. This framework links unpredictability with statistical uniformity and independence. It is built on the familiar complexity theory assumption that what cannot be computed in polynomial time in k is, for practical purposes, impossible to compute for moderate k. For more precise definitions, proofs, and further results, see [1, 18, 32] and the references given there.

How practical is this framework? Are there efficient polynomial-time perfect generators around? Which k is large enough? These questions have no clear answers at present time. Some generator families are *conjectured* to be polynomial-time perfect, and among them is the BSS generator [1] discussed in Section V, but no one has been able to *prove* even the existence of such a family. The fast generators commonly used for simulation are known not to be polynomial-time perfect. However, the "recommendable ones" seem to have good enough statistical properties.

B What is a good periodic sequence?

Choosing a fixed-size RNG amounts to choosing a deterministic and periodic sequence of (given) period length ρ. How to discriminate among such sequences? One can view the periodic sequence as a giant roulette, with ρ numbers around. Suppose U is a finite set (a discretization of $[0,1]$) and that the seed s_0 is chosen randomly, uniformly over the period length (i.e., the initial position on the roulette is chosen randomly). We would like every vector $\boldsymbol{u} = (u_0, \ldots, u_{t-1}) \in U^t$ to have the same probability, ideally for every t. This is possible only if $|U|^t$ divides ρ. For $t > \log(\rho)/\log(|U|)$, there is not enough positions around the roulette to cover U^t, so some of these vectors must have zero probability, i.e., $\Psi_t = \{\boldsymbol{u}_n = (u_n, \ldots, u_{n+t-1}),\ 1 \le n \le \rho\}$ is a strict subset of U^t. In fact, for large t, Ψ_t can cover only a tiny fraction of U^t, and the question arises of how the points of Ψ_t should be spread over U^t. Some want Ψ_t to look like a random set of points, for each t. Others prefer Ψ_t to be very uniformly spread over U^t, based on the rationale that Ψ_t should be viewed as a discretization of U^t, from which a few points are drawn randomly, and that the uniform distribution is better approximated by a regular and uniform discretization than by a random one. This makes sense only if several orders of magnitude less than ρ random numbers are used.

For example, let $U = \{0, 1/1024, 2/1024, \ldots, 1023/1024\}$, so we look at the 10 most significant bits of each output value, and $\rho = 2^{200} = 1024^{20}$. In dimensions $t \le 20$, it is possible to have perfect equidistribution, all t-dimensional vectors appearing the same number of times over the entire period. But for $t = 30$, for example, only one point out of 2^{100}, belongs to Ψ_t, in the best case. Theoretical analysis of RNGs is meant to understand the structure well-enough to make sure that the points are approximately equidistributed for small t and that Ψ_t is evenly distributed for t up to some large value such as 50, and preferably more. This will be our criterion for constructing "fair" generators. Similar ideas are used in, e.g., [5, 17, 21, 48]. Perhaps generators with well-understood structural properties look less random, but the less understood ones may hide important defects.

C Discrepancy

Well-established measures of uniformity for finite sequences of numbers are based on the notion of *discrepancy*, which is well-covered in [42]. Let $P_N = \{\boldsymbol{u}_n = (u_n, \ldots, u_{n+t-1}),\ 0 \le n \le N-1\}$ be a set of N points formed by overlapping vectors of successive values. For any box of the form $R = \prod_{j=1}^{t} [\alpha_j,\ \beta_j)$, with $0 \le \alpha_j < \beta_j \le 1$, let $I(R)$ be the number of points $\boldsymbol{u}_n$ falling into R, and $V(R) = \prod_{j=1}^{t} (\beta_j - \alpha_j)$ the volume of R. If $\mathcal{R}$ is the set of all such regions R, then

$$D_N^{(t)} = \max_{R \in \mathcal{R}} |V(R) - I(R)/N|$$

is called the *extreme discrepancy* of P_N. If we restrict $\mathcal{R}$ to only the boxes with one corner at the origin, then the corresponding quantity is called the *star discrepancy*. Other variants also exist.

Low discrepancy is linked to super-uniformity. So, in line with our previous arguments, one might want to *minimize* the discrepancy for $N = \rho$, the period length. On the other hand, for $N \ll \rho$, the discrepancy, viewed as a random variable, should behave as the discrepancy of a sequence of i.i.d. $U(0,1)$ random variables; e.g., be (roughly) in $O(N^{-1/2})$. The linear-type generators discussed in the forthcoming sections enjoy these properties, at least asymptotically [42]. Unfortunately, discrepancy is too hard to compute, except for special cases. Only loose bounds on it are available in general.

Point sets with much lower discrepancy than typical i.i.d. uniform random points are called *low-discrepancy* point sets. Infinite sequences for which P_N has low discrepancy for all large enough N are called low-discrepancy sequences. Such point sets and sequences are also called *quasi-random* and are sometimes used for finite-dimensional numerical integration in replacement of (or in combination with) the basic Monte Carlo method [42, 48].

III LINEAR METHODS BASED ON INTEGER ARITHMETIC

A The Multiple-Recursive Generator

Define the linear recurrence

$$x_n = (a_1 x_{n-1} + \cdots + a_k x_{n-k}) \bmod m, \tag{1}$$

with *order* k, *modulus* m (a positive integer), and integer *coefficients* $a_1, \ldots, a_k$ in the range $\{-(m-1), \ldots, m-1\}$. Let $\mathbb{Z}_m = \{0, 1, \ldots, m-1\}$ with operations performed modulo m. The state at step n is $s_n = (x_{n-k+1}, \ldots, x_n) \in \mathbb{Z}_m^k$ and the output function can be defined simply by $u_n = G(s_n) = x_n/m$, giving a value in $[0,1]$, or by a more refined transformation if a better resolution than $1/m$ is required. This is the *multiple recursive generator* (MRG) [17, 20, 42]. The special case where $k = 1$ is known as the *linear congruential generator* (LCG). Then, a_1 is denoted a, and a constant is sometimes added to the recurrence, yielding

$$x_n = (a x_{n-1} + c) \bmod m. \tag{2}$$

When $c = 0$, the LCG is called a *multiplicative* LCG (or MLCG).

The largest possible period for (1) is $\rho = m^k - 1$, which is reached if and only if m is prime and the characteristic polynomial of the recurrence, defined by $P(z) = z^k - a_1 z^{k-1} - \cdots - a_k$, is a primitive polynomial over $\mathbb{Z}_m$, identified as the finite field with m elements. This means that the smallest positive integer ν such that $(z^\nu \bmod P(z)) \bmod m = 1$ is $\nu = m^k - 1$. Equivalent conditions that are more convenient to verify are given in [17]. For $k > 1$, for P to be primitive, it is necessary that a_k and at least another coefficient a_j be non-zero. It is also easy to find primitive polynomials of this form, and they yield the more economic recurrence:

$$x_n = (a_r x_{n-r} + a_k x_{n-k}) \bmod m. \tag{3}$$

For faster speed, one may prefer $m = 2^e$, a power of two, instead of a prime m. The recurrence (1) or (2) is then easier to compute (e.g., if $m = 2^{32}$ on a 32-bit computer), but the period length is much smaller than $m^k - 1$. For $m = 2^e$, the maximum period length is 2^{e-2} if $k = 1$ and $e \geq 4$ (it is reached if $a_1 = 3$ or $5 \pmod 8$ and x_0 is odd [17, p.20]), and $(2^k - 1)2^{e-1}$ if $k > 1$. For example, if $k = 7$ and $m = 2^{31} - 1$, the maximal period length is $(2^{31} - 1)^7 - 1 \approx 2^{217}$, whereas if $k = 7$ and $m = 2^{31}$, the upper bound becomes $(2^7 - 1)2^{31-1} < 2^{37}$, which is more than 2^{180} times smaller! Another drawback of power-of-two moduli is that the low-order bits are typically much too regular. For $k = 1$, an upper bound on the period length of the ith least significant bit of x_n is $\max(1, 2^{i-2})$ [2], and if a full cycle is split into 2^d equal segments, then all segments are identical except for their d most significant bits. For $k > 1$, the upper bound on the period length of the ith least significant bit is $(2^k - 1)2^{i-1}$. For $k = 7$ and $m = 2^{31}$, for example, the least significant bit has period length at most $2^7 - 1 = 127$, the second least significant bit has period length at most $2(2^7 - 1) = 254$, and so on.

For $k = 1$ and $m = 2^e$, one can obtain a period length of $\rho = m$ by using (2) with $c > 0$, c odd, and $a \bmod 4 = 1$ (see [17, p.16]). However, the short periodicity of the low-order bits remains: the ith least significant bit of x_n has period length 2^i. For $k > 1$, adding a constant c in the recurrence (1) is not much helpful. The recurrence then becomes equivalent to a linear recurrence of order $k + 1$ with $c = 0$, so the period length cannot exceed $(2^{k+1} - 1)2^{e-1}$ (see [20]).

B Implementation for Prime m

For prime m, computing $ax \bmod m$ in integer arithmetic can get tricky, because the product ax is often not representable as an ordinary integer on the computer at hand (for example, if $m = 2^{31} - 1$ on a 32-bit computer). General techniques to compute this product modulo m without overflow, using only integer arithmetic, are compared in [30]. In the case when $a = i$ or $a = \lfloor m/i \rfloor$, for $i < \sqrt{m}$, there is an efficient *approximate factoring* method [27]. Another approach is to perform all the arithmetic modulo m in double precision floating-point. For example, if the floating-point numbers are represented over 64 bits and if the computations are performed according to the IEEE standard, then all nonnegative integers less or equal to 2^{53} are represented exactly in floating point. In this case, if $a_i(m-1) \le 2^{53}$, then each product $a_i x_{n-i}$ will always be represented exactly in floating point. Similarly, if $(|a_1| + \cdots + |a_k|)(m-1) \le 2^{53}$, then $a_1 x_{n-1} + \cdots + a_k x_{n-k}$ is always represented exactly. The following instructions then perform the assignment $x := ax \bmod m$:

$$y := ax; \qquad x := y - \lfloor y/m \rfloor m.$$

C Jumping ahead

To jump directly from x_n to $x_{n+\nu}$ with an MLCG, for large ν, use the relation

$$x_{n+\nu} = a^\nu x_n \bmod m = (a^\nu \bmod m) x_n \bmod m.$$

The constant $a^\nu \bmod m$ can be pre-computed using a divide-to-conquer algorithm [20]. For an LCG with $c \neq 0$, one has

$$x_{n+\nu} = \left(a^\nu x_n + \frac{c(a^\nu - 1)}{a - 1}\right) \bmod m.$$

To jump ahead with the MRG, one way is to use the fact that it can be represented as a matrix MLCG: $X_n = AX_{n-1} \bmod m$, where X_n is s_n represented as a column vector and A is a $k \times k$ square matrix [20]. Jumping ahead is then achieved in the same way as for the MLCG:

$$X_{n+\nu} = A^\nu X_n \bmod m = (A^\nu \bmod m) X_n \bmod m.$$

Another way is to transform the MRG into its polynomial representation [21], in which jumping ahead is achieved by multiplying this polynomial by $z^\nu \bmod P(z)$, and then apply the inverse transformation to recover the original representation.

D Lattice Structure of LCGs and MRGs

The set

$$T_t = \{\boldsymbol{u}_n = (u_n, \ldots, u_{n+t-1}) \mid n \geq 0,\ (x_0, \ldots, x_{k-1}) \in \mathbb{Z}_m^k\} \tag{4}$$

of all t-dimensional vectors of successive values produced by (1) and $u_n = x_n/m$, from all possible initial seeds, is the intersection of a lattice L_t with the hypercube $[0,1)^t$ (see, e.g., [17, 27, 31]).

For $t \leq k$, L_t contains all vectors in dimension t whose coordinates are multiples of $1/m$. But for $t > k$, L_t contains only a small fraction of these vectors, namely m^k out of m^t. The lattice structure of T_t means that its points lie on a limited number of equidistant parallel hyperplanes, at distance, say, d_t apart. Figures 1–2 illustrate the lattice structure of two small MLCGs with modulus $m = 251$ (a prime), in dimension 2. They have $a = 78$ and $a = 167$, respectively. The figures show all 250 pairs of successive values (u_n, u_{n+1}) produced by these (full period) generators. In dimension 2, the parallel hyperplanes are lines, and one can see from the figures that d_2 is smaller, and the points are more uniformly distributed, for $a = 78$ than for $a = 167$.

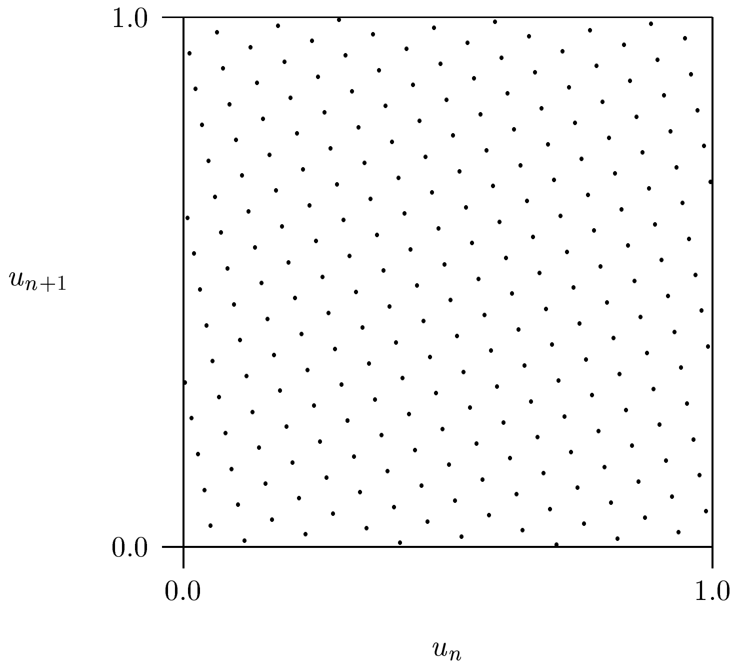

Figure 1: All pairs (u_n, u_{n+1}) for the LCG with $m = 251$ and $a = 78$

Computing d_t is often called the *spectral test*, for historical reasons [17]. The available algorithms

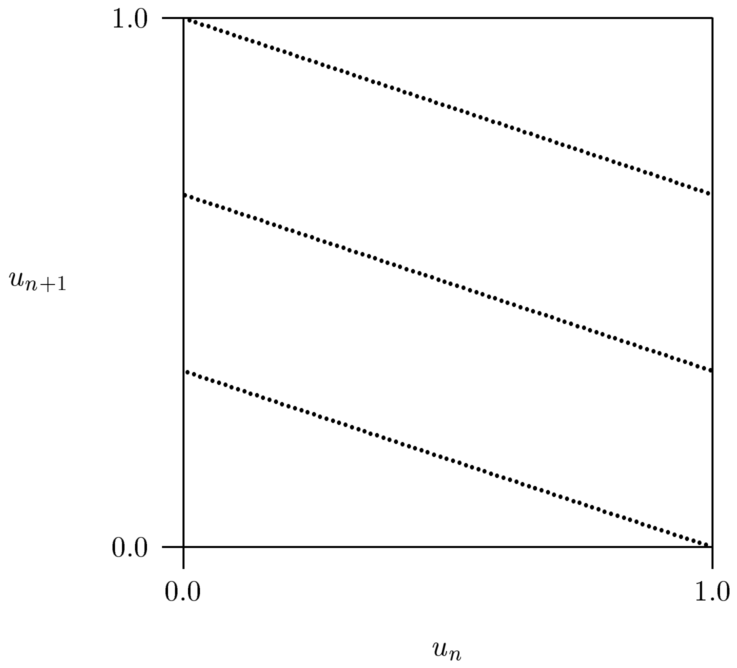

Figure 2: All pairs (u_n, u_{n+1}) for the LCG with $m = 251$ and $a = 167$

for doing so are exponential in t in the worst case, but current implementations permit one to compute d_t up to 40 or more dimensions, for m exceeding 2^{100} [31].

As we saw in the examples, a large d_t is bad, because it means thick empty slices between the hyperplanes. For any lattice, the value of d_t is nondecreasing with respect to t. An absolute lower bound, for any lattice whose intersection with $(0, 1]^t$ has m^k points, is given by

$$d_t \geq d_t^* = \frac{1}{\gamma_t m^{k/t}} \tag{5}$$

where γ_t is the Hermite constant for quadratic forms in dimension t, currently known for $t \leq 8$ (see [17]). For larger t, lower and upper bounds on γ_t are known. For example, an upper bound can be deduced from Rogers bound on the density of sphere packings [6, p.88] and [25]. The value of d_t can then be normalized to $S_t = d_t^*/d_t$, where γ_t in the definition of d_t^* is replaced by its upper bound for $t > 8$. Then, for any fixed $T > k$, one can define the figure of merit $M_T = \min_{k<t\leq T} S_t$, which lies between 0 and 1. We seek generators with large values of M_T.

Another lower bound on d_t, for $t > k$, is (see [24]): $d_t^2 \geq 1/\left(1 + \sum_{j=1}^{k} a_j^2\right)$, which implies that an MRG whose sum of squares of coefficients is small is guaranteed to be bad with respect to d_t.

Other figures of merit for random number generators, in terms of their lattice structure, have been introduced as well. For example, one can count the minimal number of hyperplanes that contain all the points, or compute the ratio of lengths of the shortest and longest vector in a Minkowski-reduced basis of the lattice. See, for example, [12, 31] and the references given there. These other figures of merit do not tell us much more than d_t.

Values of d_t and S_t for certain MRGs and LCGs can be found, e.g., in [17, 12, 27, 31, 25]. The values of S_t or M_T for different generators can be compared only if these generators have the same period length (i.e., the same lattice density). A generator with period length less than 2^{60}, for example, should not be recommended for serious use, whatever be its values of S_t, because such a period length is too short. MRGs with large periods and good figures of merit are given in [27].

E Lacunary Indices

The vectors forming the set T_t do not have to be constructed from successive output values. More generally, one can select a fixed set of *lacunary indices* $I = \{i_1, i_2, \cdots, i_t\}$ and define, for an MRG,

$$T_t(I) = \{(u_{i_1+n}, \ldots, u_{i_t+n}) \mid n \geq 0,\ (x_0, \ldots, x_{k-1}) \in \mathbb{Z}_m^k\}.$$

Let $L_t(I)$ be the lattice spanned by $T_t(I)$ and $\mathbb{Z}^t$, and let $d_t(I)$ be the distance between the hyperplanes in this lattice. This $d_t(I)$ can be computed as explained in [31]. An easy lower bound on $d_t(I)$ is given as follows ([7, 24]): If $\{i : a_{k-i} \neq 0\} \subseteq I$, then $d_t(I)^2 \geq 1/\left(1 + \sum_{i=1}^k a_i^2\right)$. In particular, if $x_n = (a_r x_{n-r} + a_k x_{n-k}) \bmod m$ and $I = \{0, k-r, k\}$, then $d_3(I) \geq (1 + a_r^2 + a_k^2)^{-1/2}$. A further special case is the so-called *lagged-Fibonacci* generator, based on a recurrence with two nonzero coefficients, $a_r = \pm 1$ and $a_k = \pm 1$. In this case, for $I = \{0, k-r, k\}$, $d_3(I) \geq 1/\sqrt{3} \approx .577$, and all the vectors (u_n, u_{n+r}, u_{n+k}) produced by this generator lie in only two planes! Specific instances of this generator are the one proposed in [17], based on the recurrence $x_n = (x_{n-24} + x_{n-55}) \bmod 2^e$ for e equal to the computer's word length, and the "`addrans`" function in the SUN Unix library, based on $x_n = (x_{n-5} + x_{n-17}) \bmod 2^{24}$. These generators should be discarded.

F Combined LCGs and MRGs

It is often advocated to combine the output of different generators to enlarge the period and improve the statistical properties [20, 36]. Combination is supported by a large amount of empirical experience. For sequences of independent random variables, certain types of combinations (e.g., addition modulo 1) yield a combined sequence whose distribution is provably closer to the uniform than that of any of its components alone [3, 36]. However, in the case of *deterministic* sequences, these results are not really meaningful and can only be used as heuristic arguments. One can construct counterexamples where the combined sequence is worse than each of its components. Therefore, to assess its quality, one must study the structural properties of the combined generator itself. This applies not only to MRGs, but more generally.

Two classes of widely used combination methods are:

(i) Shuffling one sequence with another or with itself;

(ii) Adding two or more integer sequences modulo some integer m_0, or adding sequences of real numbers in $[0,1]$ modulo 1, or adding binary fractions bitwise modulo 2.

Methods of the first class shuffle (locally) the sequence of a generator with the aid of a second generator. Usually, the first generator fills up a table and then, at each step, the output of the second generator determine a position in the table, the value at that position is returned as output of the combined generator, and is replaced in the table by the next output value from the first generator. There are several variants of this shuffling scheme. Sometimes, the same generator is used for both fill-up and selection. Shuffling has good empirical support, but its effects are not so well understood theoretically and efficient jump-ahead algorithms are not available.

The combination of MRGs by modular addition is better understood and, in that case, jumping ahead is easy, because it suffices to do it with the individual components, independently. Consider the J recurrences:

$$x_{j,n} = (a_{j,1}x_{j,n-1} + \cdots + a_{j,k}x_{j,n-k}) \bmod m_j,$$

for $j = 1, \ldots, J$. Assume that the m_j are pairwise relatively prime and that each recurrence is purely periodic (has zero transient) with period length ρ_j. Let $\delta_1, \ldots, \delta_J$ be arbitrary integers such that for each j, δ_j and m_j have no common factor. Define the combinations:

$$z_n = \left(\sum_{j=1}^{J} \delta_j x_{j,n}\right) \bmod m_1; \qquad u_n = z_n/m_1 \tag{6}$$

and

$$w_n = \left(\sum_{j=1}^{J} \frac{\delta_j x_{j,n}}{m_j}\right) \bmod 1. \tag{7}$$

Let $k = \max(k_1, \ldots, k_J)$ and $m = \prod_{j=1}^{J} m_j$. The following is proved in [33] for the case of MLCG components ($k = 1$) and in [22] for the more general case:

(i) The sequences $\{u_n\}$ and $\{w_n\}$ have period length $\rho = \mathrm{lcm}(\rho_1, \ldots, \rho_J)$ (the least common multiple of the ρ_j).

(ii) The w_n obey the recurrence:

$$x_n = (a_1 x_{n-1} + \cdots + a_k x_{n-k}) \bmod m; \qquad w_n = x_n/m, \tag{8}$$

where the a_i are computed as explained in [22] and do not depend on the δ_j.

(iii) One has $u_n = w_n + \epsilon_n$, with $\Delta^- \le \epsilon_n \le \Delta^+$, where Δ^- and Δ^+ can be computed as explained in [22] and are very small when the m_j are close to each other.

The combinations (6) and (7) can then be viewed as efficient ways to implement an MRG with large (composite) modulus m. The idea is to choose MRG components that are fast and easy to implement (e.g., only two nonzero coefficients) and for which the MRG (8) has good structural properties. If m_j is prime and $\rho_j = m_j^{k_j} - 1$ for each j, then each ρ_j is even, so $\rho \le (m_1^{k_1} - 1) \cdots (m_J^{k_J} - 1)/2^{J-1}$.

A specific combined MRG with $J = 2$, $k = 3$, and period length $\rho \approx 2^{185}$, is proposed in [22]. It was selected after a computer search for combined MRGs of this size, with good lattice structure. An implementation in the C language is given.

G Matrix MRGs

A natural extension of (1) is to define MRGs in matrix form:

$$X_n = (A_1 X_{n-1} + \cdots + A_k X_{n-k}) \bmod m \tag{9}$$

where $A_1, \ldots, A_k$ are $L \times L$ matrices and each X_n is a L-dimensional vector with components in $\mathbb{Z}_m$. At each step, one can use X_n to produce an output *vector*. Niederreiter [45] studies this generalization and calls it the *multiple recursive matrix method*.

H Linear Recurrences with Carry

Consider the recurrence:

$$
\begin{aligned}
x_n &= (a_1 x_{n-1} + \cdots + a_k x_{n-k} + c_{n-1}) \bmod b, & (10)\\
c_n &= (a_1 x_{n-1} + \cdots + a_k x_{n-k} + c_{n-1}) \operatorname{div} b, & (11)\\
u_n &= x_n / b.
\end{aligned}
$$

where div mean *integer division*. This is similar to (1), except for the *carry* c_n, and the state is now $s_n = (x_n, \ldots, x_{n+k-1}, c_n)$. This type of generator, proposed by G. Marsaglia, is studied in [8], and called *Multiply-with-Carry* (MWC). Define $a_0 = -1$, $m = \sum_{\ell=0}^{k} a_\ell b^\ell$, and let a be the inverse of b in arithmetic modulo m, assuming for now that $m > 0$. If we define

$$w_n = \sum_{i=1}^{\infty} x_{n+i-1} b^{-i}, \tag{12}$$

then the sequence $\{w_n\}$ obeys (see [8]):

$$z_n = a z_{n-1} \bmod m; \qquad w_n = z_n / m. \tag{13}$$

The MWC is then equivalent to this LCG, up to precision $1/b$. A period length of $(m-1)/2$ can be achieved if b is a power of two, and this is the major advantage over an MRG. For example, one may take $b = 2^{32}$. Note that the MWC has much more states than $(m-1)/2$, some of them being transient. The set of recurrent states is precisely characterized in [8]. In particular, if $a_\ell \geq 0$ for $\ell \geq 1$, then the carry c_n is always smaller than $a_1 + \cdots + a_k$, so if $b-1$ fits into a computer word and $a_1 + \cdots + a_k \leq b$, then each component of the state fits into a computer word.

The lattice structure of the LCG (13) satisfies the following (see [8]): for $t \leq k$, d_t does not depend on $a_1, \ldots, a_k$, but only on b, and for $t = k+1$, one has

$$d_{k+1} = (1 + a_1^2 + \cdots + a_k^2)^{-1/2}. \tag{14}$$

To minimize this distance and get a large m, one should choose a_k close to b, with $a_0 + \cdots + a_k \leq b$. Specific parameter sets are proposed in [37, 8].

Special cases of the MWC, called the add-with-carry and subtract-with-borrow generators, were proposed in [39] and analyzed in [7, 50]. These special cases have $a_r = \pm 1$, $a_k \pm 1$, and the other a_ℓ

equal to 0. They resemble the lagged-Fibonacci generators and turn out to have the same defect: $d_{k+1} = 1/\sqrt{3}$ from (14), and all vectors of the form (w_n, w_{n+r}, w_{n+k}) lie in only two planes in the three-dimensional unit cube [7].

One way to get around this problem is to take only blocks of k successive values of x_n, and skip many values in between, so that triplets of the form (w_n, w_{n+r}, w_{n+k}) never appear. Such an approach is proposed in [35], for a specific generator, with heuristic justification based on Lyapunov exponents, and a Fortran implementation of the generator is given in [16].

IV THE DIGITAL METHOD

A Digital Multistep Sequences

To improve the resolution of the output, one can construct each u_n from several successive x_n in (1), as follows. Redefine

$$u_n = \sum_{j=1}^{L} x_{ns+j-1} m^{-j}. \qquad (15)$$

where s is called the *step size* and L is the *number of digits* in the expansion. The u_n are now multiples of m^{-L} and this sequence $\{u_n\}$ is called a *digital multistep sequence* [21, 42]. If the MRG sequence has period ρ and if s is relatively prime with ρ, then $\{u_n\}$ also has period ρ.

B LFSR or Tausworthe Generators

An important special case is $m = 2$. Then, $\{x_n\}$ becomes a sequence of bits and the corresponding generator is called a *Linear Feedback Shift Register* (LFSR) or *Tausworthe* generator [21, 42, 47] (often with the bits of u_n filled in reverse order than in (15)). Specific implementations for recurrences of the form $x_n = (x_{n-r} + x_{n-k}) \bmod 2$, with $s \le r$ and $2r > k$, are given in [23, 49, 48]. For jump-ahead algorithms, see [23].

Simple LFSRs with only two nonzero coefficients have bad structural properties (see, e.g., [4, 23, 41, 48] and other references there), but several of them can be combined, much as for the combined MRGs, yielding fast generators with good properties. Take J LFSR recurrences, the jth one having a primitive characteristic polynomial $P_j(z)$ of degree k_j, and step size s_j relatively prime with the period length $\rho_j = 2^{k_j} - 1$. Assume that the polynomials $P_j(z)$ are pairwise relatively prime, that the ρ_j are also relatively prime, that these LFSRs use a common L, and let u_n be the bitwise addition modulo 2 of the output values of these J LFSR components at step n. Then, this combined sequence $\{u_n\}$ is exactly equivalent to that of a LFSR generator with characteristic polynomial $P(z) = P_1(z) \cdots P_J(z)$, whose period length is $\rho = \prod_{j=1}^{J} \rho_j$. See [49, 23, 52] for more details and specific implementations. For better statistical behavior, it is recommended that $P(z)$ have approximately half of its coefficients equal to one.

C Matrix Methods and GFSR Generators

The digital method can be applied also to the matrix version (9) in the following way:

$$u_n = \sum_{j=1}^{L} x_{n,j} m^{-j}. \tag{16}$$

where $x_{n,1}, \ldots, x_{n,L}$ are the first L components of X_n. This is studied in [43]. Under specific conditions on the seeds (see [13, 21]), (16) gives exactly the same sequence as (15) with a proper choice of s, so these are then two different implementations of the same generator. The implementation (9) and (16) needs more memory but may be faster. An important instance of it is the Generalized Feedback Shift Register (GFSR) generator [14, 34, 51], for which $m = 2$ and L is the computer's word length, so (9) can then be computed by a bitwise exclusive-or of the X_{n-j} for which $a_j = 1$. If there are only two nonzero coefficients, say a_k and a_r, we obtain

$$X_n = X_{n-r} \oplus X_{n-k},$$

where $\oplus$ denotes the bitwise exclusive-or. This gives a very fast generator, but our caveat for trinomial-based LFSR generators also applies here.

The GFSR requires kL bits of memory, but its period length is only $2^k - 1$ instead of near 2^{kL} as we should expect. It then comes to mind to *twist* it slightly, to improve its properties without impairing the speed too much. For example, as a coefficient of X_{n-k}, one can take some binary matrix A_k instead of the identity matrix, and choose A_k so that the period becomes $2^{kL} - 1$ and the implementation remains fast. A specific way of doing this and a concrete implementation are given in [40].

D Equidistribution Properties for the Digital Method

A partition of the hypercube $[0,1)^t$ into $m^{t\ell}$ cubic cells of equal size is called a (t,ℓ)-*equidissection in base* m. A set of points is (t,ℓ)-*equidistributed* if each of these cells contains the same number of points. For a set of m^k points, this is possible only for $\ell \leq \lfloor k/t \rfloor$. For a given digital multistep sequence, let

$$T_t = \{\boldsymbol{u}_n = (u_n, \ldots, u_{n+t-1}) \ : \ n \geq 0, \, (x_0, \ldots, x_{k-1}) \in \mathbb{Z}_m^k\} \tag{17}$$

and $\ell_t = \min(L, \lfloor k/t \rfloor)$. If T_t is (t, ℓ_t)-equidistributed for all $t \leq k$, we say that the generator is *maximally equidistributed* (ME). We also call it collision-free (CF) if for all t, for $\ell_t < \ell \leq L$, no cell contains more than one point. ME-CF generators have their point sets evenly distributed in the hypercube in all dimensions.

GFSRs are (t, L)-equidistributed for $t = \lfloor k/L \rfloor$ if and only if their initial state satisfies independence conditions given in [14]. One can verify (t,ℓ)-equidistribution rapidly by computing the rank of a binary matrix that expresses the first ℓ bits of $\boldsymbol{u}_n$ as a linear combination of $(x_0, \ldots, x_{k-1})$. Reference [23] explains this, and provides specific ME and ME-CF combined LFSR generators, with an implementation in C.

A stronger notion than (t,ℓ)-equidistribution is that of a (q,k,t)-*net in base* m, where the partition into cubic boxes is replaced by a richer set of partitions into rectangular boxes of equal volume m^{q-k}, with each coordinate of each box corner equal to a multiple of $1/m$. Niederreiter [42] defines a figure of merit $r^{(t)}$ such that for $t > \lfloor k/L \rfloor$, the m^k points of T_t for (15) form a (q,k,t)-net in base m with $q = k - r^{(t)}$. This $r^{(t)}$ is difficult to compute for medium and large t (say, $t > 8$).

V NONLINEAR METHODS

The generators discussed so far have a linear structure, which may be deemed too regular. To get away from it, one can either use a nonlinear transition function T, or keep T linear and use a nonlinear transformation G to produce the output. Several types of nonlinear generators have been proposed and analyzed in the literature; see, e.g., [10, 11, 15, 18, 32, 42, 44] and other references given there. The sequences produced by these generators, over their entire periods, tend to behave more like truly random sequences than like superuniform ones (e.g., in terms of their discrepancy bounds). A major drawback of nonlinear generators is that they are typically significantly slower than the linear ones. A general class of nonlinear transformations is defined by:

$$x_n = T(x_{n-1}) = f(x_{n-1}) \bmod m, \qquad x_{n-1} \in \mathbb{Z}_m, \tag{18}$$

where m is a positive integer. The output can be produced simply by $u_n = x_n/m$, or by more complicated means if desired.

A Inversive Congruential Generators

Some generators use inversion modulo m as their nonlinear transformation. For example, let $\{x_n\}$ be an MRG sequence with prime m and define the output as

$$z_n = (\tilde{x}_{n+1}\tilde{x}_n^{-1}) \bmod m \qquad \text{and} \qquad u_n = z_n/m, \tag{19}$$

where $\tilde{x}_i$ denotes the ith nonzero value in the sequence $\{x_n\}$, and $\tilde{x}_n^{-1} = \tilde{x}_n^{m-2} \bmod m$ is the inverse of $\tilde{x}_n$ modulo m, which can be computed with $O(\log m)$ multiplications modulo m. The sequence $\{z_n\}$ has period m^{k-1}, under conditions given in [10, 42].

Another approach is the *explicit inversive congruential* method: Let $x_n = an + c$ for $n \geq 0$, where $a \neq 0$ and c are in $\mathbb{Z}_m$ and m is prime, and define

$$z_n = x_n^{-1} = (an+c)^{m-2} \bmod m \qquad \text{and} \qquad u_n = z_n/m. \tag{20}$$

This sequence has period $\rho = m$ and, according to [11], seems to enjoy the most favorable properties among the currently proposed inversive families.

Inversive congruential generators with power-of-two moduli have been studied, but their structural properties are not as good as for the generators with prime moduli [11, 28]. Inversive generators can be combined much in the same way as MRGs and LFSRs, e.g., by addition modulo 1.

B The BBS and Other Cryptographic Generators

Several types of nonlinear generators, some conjectured to be polynomial-time perfect, have been proposed in the area of cryptology [18, 32]. A well-known example is the BBS generator [1], defined as follows. For an instance of size k, the state is a triplet (p, q, x) where p and q are $(k/2)$-bit primes, both congruent to 1 modulo 4, and x is a quadratic residue modulo $m = pq$, relatively prime to m. The initial state is random, but after it is selected, only x changes, according to the recurrence: $x_n = x_{n-1}^2 \bmod m$. At each step, the generator outputs the ν_k least significant bits of x_n, where $\nu_k = O(\log k)$. Under the assumption that there is no general polynomial-time (in k) algorithm to find p or q given m, the BBS generator has been proved polynomial-time perfect. Of course, all of this is only asymptotic. In practice, how large k should be and how small ν_k should be? This remains unclear. This generator is rather slow compared with the linear ones, but for certain cryptographic applications, it is a good choice.

VI EMPIRICAL STATISTICAL TESTING

Define the null hypothesis $\mathcal{H}_0$: "$u_1, u_2, u_3, \ldots$ are i.i.d. $U(0,1)$ random variables". Formally, we know that this hypothesis is false, but can we detect it? A test is defined by a statistic T, function of a fixed set of u_i, whose distribution under $\mathcal{H}_0$ is known. Different tests may detect different defects. Ideally, T should resemble the random variable of practical interest, so that a bad interference between the generator's structure and the problem of interest would show up in the test. But designing specific statistical tests for each problem is not practical. For general purpose random number generators in software libraries, the best we can do is to apply a large and varied set of tests. Note that no statistical test can prove that a generator is totally foolproof. For better confidence, it is a good idea to re-run important simulations with random number generators of totally different types.

A standard (one-level) test will reject $\mathcal{H}_0$ if T is too far in the tail of its theoretical distribution. A two-level test would compute N "independent" copies of T and compare their empirical distribution with the theoretical distribution of T under $\mathcal{H}_0$, via a goodness-of-fit test such as those of Kolmogorov-Smirnov or Anderson-Darling [17, 46]. Several statistical tests for random number generators are given in [17, 21, 29, 28, 36] and other references given there.

Some experience with empirical testing tells us the following: (i) Generators with period shorter than, say 2^{32}, easily fail a lot of tests and should no longer be used; (ii) Generators with power-of-two moduli are easier to crack than those with prime moduli, especially when looking at lower-order bits; (iii) LFSRs and GFSRs based on primitive trinomials, or lagged-Fibonacci and AWC/SWB generators, are also easy to crack. (iv) Combined generators with long period and good lattice structure or good equidistribution do well in the tests. Generators based on more complicated recurrences tend to do better. This favors combined generators.

VII AVAILABLE IMPLEMENTATIONS

We conclude by giving pointers to available implementations. No generator can be guaranteed against all possible defects, but those that we now mention have fairly good theoretical support, have been extensively tested, and are easy to use. Our list includes the twisted GFSR in [40], the combined MRG in [22], the combined LCG in [26], the combined Tausworthe generator in [23], the MRG in [27], and the RANLUX code in [16].

For some applications (e.g., for implementing certain variance reduction methods in simulation), it is convenient to have several (virtual) generators available, each one having its sequence partitioned into very long segments, and where one can jump rapidly to the beginning of the next segment, or the previous segment, or the first segment, for each generator. One such software package is described in [26].

RECOMMENDED READINGS FOR FURTHER STUDY

For the readers interested in learning more about the design of pseudorandom number generators, we recommend the references [17, 21, 42]. Several articles and other material related to RNGs are available from the web pages:

`http://www.iro.umontreal.ca/~lecuyer`

`http://random.mat.sbg.ac.at`

ACKNOWLEDGMENTS

This work has been supported by the National Science and Engineering Research Council of Canada grants # ODGP0110050 and SMF0169893, and FCAR-Québec grant # 93ER1654.

REFERENCES

1. L. Blum, M. Blum, and M. Schub. A simple unpredictable pseudo-random number generator. *SIAM Journal on Computing*, 15(2):364–383, 1986.
2. P. Bratley, B. L. Fox, and L. E. Schrage. *A Guide to Simulation*. Springer-Verlag, New York, second edition, 1987.
3. M. Brown and H. Solomon. On combining pseudorandom number generators. *Annals of Statistics*, 1:691–695, 1979.
4. A. Compagner. The hierarchy of correlations in random binary sequences. *Journal of Statistical Physics*, 63:883–896, 1991.
5. A. Compagner. Operational conditions for random number generation. *Physical Review E*, 52(5-B):5634–5645, 1995.
6. J. H. Conway and N. J. A. Sloane. *Sphere Packings, Lattices and Groups*. Grundlehren der Mathematischen Wissenschaften 290. Springer-Verlag, New York, 1988.
7. R. Couture and P. L'Ecuyer. On the lattice structure of certain linear congruential sequences related to AWC/SWB generators. *Mathematics of Computation*, 62(206):798–808, 1994.
8. R. Couture and P. L'Ecuyer. Distribution properties of multiply-with-carry random number generators. *Mathematics of Computation*, 1997. To appear.
9. L. Devroye. *Non-Uniform Random Variate Generation*. Springer-Verlag, New York, 1986.
10. J. Eichenauer-Herrmann. Inversive congruential pseudorandom numbers: A tutorial. *International Statistical Reviews*, 60:167–176, 1992.
11. J. Eichenauer-Herrmann. Pseudorandom number generation by nonlinear methods. *International Statistical Reviews*, 63:247–255, 1995.
12. G. S. Fishman. *Monte Carlo: Concepts, Algorithms, and Applications*. Springer Series in Operations Research. Springer-Verlag, New York, 1996.
13. M. Fushimi. An equivalence relation between Tausworthe and GFSR sequences and applications. *Applied Mathematics Letters*, 2(2):135–137, 1989.
14. M. Fushimi and S. Tezuka. The k-distribution of generalized feedback shift register pseudorandom numbers. *Communications of the ACM*, 26(7):516–523, 1983.
15. P. Hellekalek. Inversive pseudorandom number generators: Concepts, results, and links. In C. Alexopoulos, K. Kang, W. R. Lilegdon, and D. Goldsman, editors, *Proceedings of the 1995 Winter Simulation Conference*, pages 255–262. IEEE Press, 1995.

16. F. James. RANLUX: A Fortran implementation of the high-quality pseudorandom number generator of Lüscher. *Computer Physics Communications*, 79:111–114, 1994.

17. D. E. Knuth. *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*. Addison-Wesley, Reading, Mass., second edition, 1981.

18. J. C. Lagarias. Pseudorandom numbers. *Statistical Science*, 8(1):31–39, 1993.

19. A. M. Law and W. D. Kelton. *Simulation Modeling and Analysis*. McGraw-Hill, New York, second edition, 1991.

20. P. L'Ecuyer. Random numbers for simulation. *Communications of the ACM*, 33(10):85–97, 1990.

21. P. L'Ecuyer. Uniform random number generation. *Annals of Operations Research*, 53:77–120, 1994.

22. P. L'Ecuyer. Combined multiple recursive generators. *Operations Research*, 44(5):816–822, 1996.

23. P. L'Ecuyer. Maximally equidistributed combined Tausworthe generators. *Mathematics of Computation*, 65(213):203–213, 1996.

24. P. L'Ecuyer. Bad lattice structures for vectors of non-successive values produced by some linear recurrences. *INFORMS Journal on Computing*, 9, 1997. To appear.

25. P. L'Ecuyer. A table of linear congruential generators of different sizes and good lattice structure. Manuscript, 1997.

26. P. L'Ecuyer and T. Andres. A random number generator based on the combination of four LCGs. Submitted, 1996.

27. P. L'Ecuyer, F. Blouin, and R. Couture. A search for good multiple recursive random number generators. *ACM Transactions on Modeling and Computer Simulation*, 3(2):87–98, 1993.

28. P. L'Ecuyer, A. Compagner, and J.-F. Cordeau. Entropy-based tests for random number generators. Submitted, 1996.

29. P. L'Ecuyer, J.-F. Cordeau, and R. Simard. Close-point spatial tests for random number generators. Submitted, 1997.

30. P. L'Ecuyer and S. Côté. Implementing a random number package with splitting facilities. *ACM Transactions on Mathematical Software*, 17(1):98–111, 1991.

31. P. L'Ecuyer and R. Couture. An implementation of the lattice and spectral tests for multiple recursive linear random number generators. *INFORMS Journal on Computing*, 1997. To

appear.

32. P. L'Ecuyer and R. Proulx. About polynomial-time "unpredictable" generators. In *Proceedings of the 1989 Winter Simulation Conference*, pages 467–476. IEEE Press, Dec 1989.

33. P. L'Ecuyer and S. Tezuka. Structural properties for two classes of combined random number generators. *Mathematics of Computation*, 57(196):735–746, 1991.

34. T. G. Lewis and W. H. Payne. Generalized feedback shift register pseudorandom number algorithm. *Journal of the ACM*, 20(3):456–468, 1973.

35. M. Lüscher. A portable high-quality random number generator for lattice field theory simulations. *Computer Physics Communications*, 79:100–110, 1994.

36. G. Marsaglia. A current view of random number generators. In *Computer Science and Statistics, Sixteenth Symposium on the Interface*, pages 3–10, North-Holland, Amsterdam, 1985. Elsevier Science Publishers.

37. G. Marsaglia. Yet another rng. Posted to the electronic billboard `sci.stat.math`, August 1, 1994.

38. G. Marsaglia. The Marsaglia random number CDROM. See `http://stat.fsu.edu/~geo/`, 1996.

39. G. Marsaglia and A. Zaman. A new class of random number generators. *The Annals of Applied Probability*, 1:462–480, 1991.

40. M. Matsumoto and Y. Kurita. Twisted GFSR generators II. *ACM Transactions on Modeling and Computer Simulation*, 4(3):254–266, 1994.

41. M. Matsumoto and Y. Kurita. Strong deviations from randomness in m-sequences based on trinomials. *ACM Transactions on Modeling and Computer Simulation*, 6(2):99–106, 1996.

42. H. Niederreiter. *Random Number Generation and Quasi-Monte Carlo Methods*, volume 63 of *SIAM CBMS-NSF Regional Conference Series in Applied Mathematics*. SIAM, Philadelphia, 1992.

43. H. Niederreiter. The multiple-recursive matrix method for pseudorandom number generation. *Finite Fields and their Applications*, 1:3–30, 1995.

44. H. Niederreiter. New developments in uniform pseudorandom number and vector generation. In H. Niederreiter and P. J.-S. Shiue, editors, *Monte Carlo and Quasi-Monte Carlo Methods in Scientific Computing*, number 106 in Lecture Notes in Statistics, pages 87–120. Springer-Verlag, 1995.

45. H. Niederreiter. Pseudorandom vector generation by the multiple-recursive matrix method. *Mathematics of Computation*, 64(209):279–294, 1995.
46. M. S. Stephens. Tests based on EDF statistics. In R. B. D'Agostino and M. S. Stephens, editors, *Goodness-of-Fit Techniques*. Marcel Dekker, New York and Basel, 1986.
47. R. C. Tausworthe. Random numbers generated by linear recurrence modulo two. *Mathematics of Computation*, 19:201–209, 1965.
48. S. Tezuka. *Uniform Random Numbers: Theory and Practice*. Kluwer Academic Publishers, Norwell, Mass., 1995.
49. S. Tezuka and P. L'Ecuyer. Efficient and portable combined Tausworthe random number generators. *ACM Transactions on Modeling and Computer Simulation*, 1(2):99–112, 1991.
50. S. Tezuka, P. L'Ecuyer, and R. Couture. On the add-with-carry and subtract-with-borrow random number generators. *ACM Transactions of Modeling and Computer Simulation*, 3(4):315–331, 1994.
51. J. P. R. Tootill, W. D. Robinson, and D. J. Eagle. An asymptotically random Tausworthe sequence. *Journal of the ACM*, 20:469–481, 1973.
52. D. Wang and A. Compagner. On the use of reducible polynomials as random number generators. *Mathematics of Computation*, 60:363–374, 1993.